Automated design of networks of Transport-Triggered Architecture processors using Dynamic Dataflow Programs

H. Yviquel[a], J. Boutellier[c], M. Raulet[b], E. Casseau[a]

[a] *University of Rennes 1, IRISA, Inria, France.*
[b] *INSA of Rennes, IETR, France.*
[c] *CSE department, University of Oulu, Finland.*

Abstract

Modern embedded systems show a clear trend towards the use of Multiprocessor System-on-Chip (MPSoC) architectures in order to handle the performance and power consumption constraints. However, the design and validation of dedicated MPSoCs is an extremely hard and expensive task due to their complexity. Thus, the development of automated design processes are of highest importance to satisfy the time-to-market pressure of embedded systems.

This paper proposes an automated co-design flow based on the high-level language-based approach of the Reconfigurable Video Coding framework. The designer provides the application description in the RVC-CAL dataflow language, after which the presented co-design flow automatically generates a network of heterogeneous processors that can be synthesized on FPGA chips. The synthesized processors are Very Long Instruction Word -style processors. Such a methodology permits the rapid design of a many-core signal processing system which can take advantage of all levels of parallelism.

The toolchain functionality has been demonstrated by synthesizing an MPEG-4 Simple Profile video decoder to two different FPGA boards. The decoder is realized into 18 processors that decode QCIF resolution video at 45 frames per second on a 50MHz FPGA clock frequency. The results show that the given application can take advantage of every level of parallelism.

Keywords: Co-design, RVC, Dataflow programs, MPSoC, TTA, FPGA

1. Introduction

Over the past few years, the use of multimedia applications in embedded systems has massively grown thanks to the commercial success of devices such

Email addresses: herve.yviquel@irisa.fr (H. Yviquel), jani.boutellier@ee.oulu.fi (J. Boutellier), mickael.raulet@insa-rennes.fr (M. Raulet), emmanuel.casseau@irisa.fr (E. Casseau)

as smartphones and tablets. The inefficiency, in terms of power consumption, of General Purpose Processors (GPP) to execute multimedia applications has already been shown [10].

As a remedy to the inefficiency of GPPs, Multiprocessor System-on-Chips (MPSoCs) have been used in the embedded domain for some years already [24]. However, the design and validation of dedicated MPSoCs is an extremely hard and expensive tasks, even more when the MPSoCs contain heterogeneous cores. Thus, the development of automated design processes, such as the one proposed in this paper, are of highest importance for the embedded industry.

Reconfigurable Video Coding (RVC) is an MPEG initiative to provide a development framework dedicated to produce and maintain video coding tools in a modular and reusable fashion [2]. The framework is based on the dataflow programming paradigm which enables reuse and reconfiguration of the coding tools. Thanks to the explicit parallelism and modularity within dataflow, the framework is ideal for automated generation of efficient heterogeneous platforms.

This paper proposes an automated co-design flow that exploits the high-level language-based approach of the RVC framework. The designer provides the application description in the RVC-CAL dataflow language [6] , after which the presented co-design flow automatically generates a network of heterogeneous Very Long Instruction Word -style processors that can be synthesized on FPGA chips and execute the application. The methodology permits the rapid design of complex many-core signal processing systems and the automation of the design flow enables easy and less error-prone development.

This paper extends preliminary work [3] in the following ways: the automatically generated code has clearly higher performance than in the earlier work; the compilation flow has been improved by the use of a better intermediate representation; the design flow is described more formally and in higher detail.

The main contributions of this paper are:

- The description of a fully automated co-design flow for instantiation of a network of heterogeneous processors from an application description based on the dataflow programming paradigm.
- The use of a new intermediate representation (IR) of the software code in order to increase the flexibility of the co-design flow. The adoption of a new IR required designing several sophisticated software transformations that are described in Section 4.2.
- A simulation infrastructure that eases the performance analysis, debugging and system integration of the proposed design. This part is explained in Section 4.3.

The paper is organized as follows. Section 2 introduces the RVC framework, the LLVM intermediate representation and the Transport-Trigger Architecture and their benefits when used in an MPSoC design flow. Related work on MPSoC design automation is presented in Section 3. Section 4 presents a precise description of the proposed design flow. Section 5 illustrates the design flow

functionality by synthesizing an MPEG-4 Simple Profile video decoder to two different FPGA boards.

2. Background

The design methodology proposed in this paper is based on several existing technologies which are briefly described below to show their benefits when used in an MPSoC design flow.

2.1. Reconfigurable Video Coding

RVC is an MPEG initiative to provide a development framework dedicated to produce and maintain video coding tools in a modular and reusable fashion [2]. The framework is based on a dataflow programming language called RVC-CAL [6] which permits the description of an application by a dataflow graph of interconnected components, called actors. The dataflow description enables reuse of the components and dynamic reconfiguration of the application. The explicit parallelism and modularity of RVC dataflow programs is ideal for automatic generation of efficient heterogeneous platforms.

RVC-CAL is a Domain-Specific Language (DSL) [22] created to help the development of signal processing systems. RVC-CAL is based on Dataflow Process Networks (DPN) [18], a special case of Kahn Process Networks (KPN) [16]. These dataflow Models of Computation (MoC) are called dynamic, because their components, called actors, can have data-dependent behavior. In other words, the behavior of an actor can depend of its input data.

This DSL is supported by the Open RVC-CAL Compiler (Orcc) [1], an open-source framework able to generate both hardware [21] and software [23] descriptions from one RVC-CAL description of an application. Orcc is based on Model-Driven Engineering (MDE) technologies [8] which speed up the design process by automating time-consuming and error-prone tasks. The Orcc project also contains the Just-in-time Adaptive Decoder Engine (Jade) [9], a software implementation of a virtual machine -based universal decoder engine.

2.2. LLVM intermediate representation

Contrary to the previous work [3], the intermediate representation (IR) used in our compilation flow is the one developed for the LLVM project (Low-Level Virtual Machine) [17]. The new IR was adopted because of its potential to carry additional information for the compiler via metadata. The potential of metadata for adaptive compilation of actors has already been shown in Jade [9]. Moreover, the LLVM IR provides the flexibility to handle the bit-accurate word lenghts of the RVC-CAL programming language.

The LLVM project is an open-source compilation framework which reaches the performance of industrial compilers while maintaining modularity and reusability. As a consequence, it has been widely used in both academia and industry. LLVM provides type safety, low-level operations, flexibility and permits the

proper representation of (practically) all high-level languages. It is used during all software-targeting phases of the presented design process.

The LLVM representation was developed to be used in three different contexts: in a classical compiler as an easily analyzable and transformable intermediate representation (IR); as a Just-In-Time compiler for fast loading from an on-disk bitcode; and finally as a human-readable assembly language representation.

2.3. Transport-Triggered Architecture

The instruction processor technology used in our design flow is Transport Triggered Architecture (TTA). TTA was chosen for the following reasons:

- **Instruction-Level Parallelism:** TTA processors are able to take advantage of the only type of parallelism which is not inherent in RVC-CAL. TTA processors resemble Very Long Instruction Word processors (VLIW) in the sense that they fetch and execute multiple instructions each clock cycle. A major difference, however, is that TTA processors have only one instruction: *move*, which simply transfers data from a processor internal place to another. For example, one move instruction can initiate a data transfer from the output of an *add* execution unit of the TTA processor to one of the inputs of a multiplier execution unit. Here, the concept *execution unit* is used in the sense of functional unit included in most of the processors. The concept of*functional unit* could be confusing because it is another name for an actor in RVC.

- **Embedded processors:** TTA processor are ideal for targeting embedded systems. In [5] it is stated that direct programming of the data transports reduces the register file traffic when compared to VLIWs, but on the other hand makes the compiler design quite challenging, as it is the compiler that schedules the data transports and makes sure conflicts are avoided. Since the compiler makes these decisions at design time, the run-time system is simplified and hence there are savings on the processor gate count and energy consumption.

- **Flexible architecture:** TTA processors are extremely configurable. The designer can make the processor tiny and energy-efficient or, if needed, increase the instruction-level parallelism of the processor arbitrarily. The TTA design environment also allows the creation of custom instructions and custom execution units, which increase the processor efficiency at the cost of making the processor somewhat more application-specific. Figure 1 shows a small example of a TTA processor composed of two buses, two execution units, one register file, one load/store unit (to manage RAM accesses) and one control unit connected to the instruction memory (ROM).

- **Robust tools**: The open source TTA Co-design Environment (TCE) [7] offers a robust toolset for the design and use of TTA processors. The TCE toolset enables the design of custom processors and their realization

into VHDL files and memory images for easy FPGA synthesis. The TCE toolset is composed of a compiler which is based on LLVM [17]. It also contains a processor simulator which permit the profiling of the executed application.

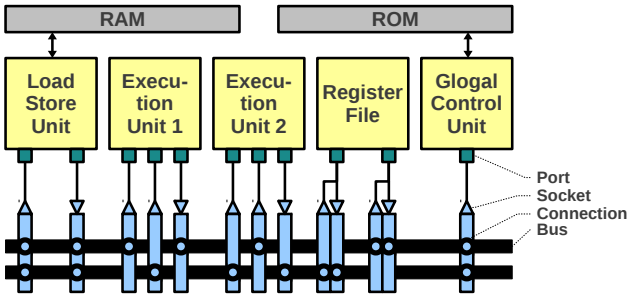

Figure 1: A simple TTA processor

3. Related work on MPSoC design flows

Park et al. classify MPSoC design approaches for signal processing systems in [20] as follows: a) the use of model-based programming languages in order to express parallelism explicitly, b) using compilation techniques to extract the parallelism from the source code and c) extending programming languages to explicitly express parallel parts of the algorithm. Our methodology consists of a mixture of a) and b): the RVC-CAL language provides data-level parallelism on the high level, whereas the TTA compiler automatically extracts the instruction-level parallelism on the low level (inside dataflow actors).

3.1. Dataflow-based approaches

Design flows from RVC applications to hardware platforms (in the sense of ASICs) have been implemented by two different tools: Openforge presented in [13] and Orcc presented in [21]. The basic idea of these approaches is the direct transformation of RVC-CAL descriptions into Register Transfer Level (RTL) descriptions suitable for FPGA or ASIC synthesis. The major difference between both methodologies comes from the abstraction level of the generated code: Openforge generates low-level and optimized HDL code dedicated to a specific platform (*close-to-gate* RTL), whereas Orcc generates high-level, portable and readable HDL code (*close-to-hand-written* RTL). Both approaches obtain excellent results in terms of gate count and frame rate. However, both of these methodologies suffer from a severe limitation as they are only applicable on single-rate RVC-CAL programs, i.e. actors can only read and write single tokens at once. However, [14] describes a way to handle this limitation using an automated transformation from multi-rate RVC-CAL programs to a single-rate

ones. Nevertheless, the results of both RTL-producing approaches show an explosion in the logical gate count and a significant reduction in the maximum frequency of the designs due to the complexity of the resulting code.

In [11], the authors present an architecture dedicated to the RVC methodology, composed of a set of predefined hardware components (ASIC) and an ARM processor. An actor is mapped to a hardware component if a suitable ASIC is available. Otherwise, the ARM processor executes the software description of the actor. The authors do not use any single high-level language description but resort to traditional software and hardware descriptions of the components (mostly in C and VHDL). The use of predefined ASIC components is a considerable limitation in terms of future evolution of the platform.

3.2. Compiler-based approaches

In [4], the authors presents a toolset which aims at parallelizing C applications for MPSoC platform. However, the process is not automated and needs the assistance of the programmer. The method is limited to thread-level parallelism.

A manually designed TTA processor for Inverse Discrete Cosine Transform (IDCT) for a video decoder is described in [19]. As our results also show, a VLIW-like processor can easily take advantage of the instruction-level parallelism of the IDCT algorithm. The authors present a real-time framerate for a 720p sequence with a clock frequency of 200MHz, but the design only encompasses a single algorithm. Moreover, the authors of [19] do not present any results about the quantity of work of manually designing such a dedicated processor.

3.3. Language-extension approaches

In [12], the authors present a multicore TTA co-design flow for parallel programming languages OpenCL and OpenMP. Contrary to our model, the processor cores are interconnected using a shared memory and exploit mechanisms such as threads and synchronization.

A parallel programming model, called embedded Message Passing Interface (eMPI), is used in [15] to establish a complete MPSoC co-design methodology using distributed memory Network on Chips (NoC) from . However, this model is based on processes and network mechanism which results of an enormous overhead according to the fine granularity of our actors.

4. Proposed TTA-based MPSoC design flow

This section presents the automated process for generating TTA-based MPSoCs from RVC application descriptions and is organized as follows: first, the hardware design flow for generating the MPSoC HDL description is presented; then, the flow of compiling the executables for each processor is described. Finally, a description of the testing infrastructure is given.

4.1. Hardware design flow

The design approach illustrated in Figure 2 shows a direct mapping of the RVC application to a hardware network of TTA processors. Each part in the application dataflow graph is mapped to an equivalent hardware component. For example, an actor is associated to a processor and a connection between two actors is replaced by a hardware FIFO channel.

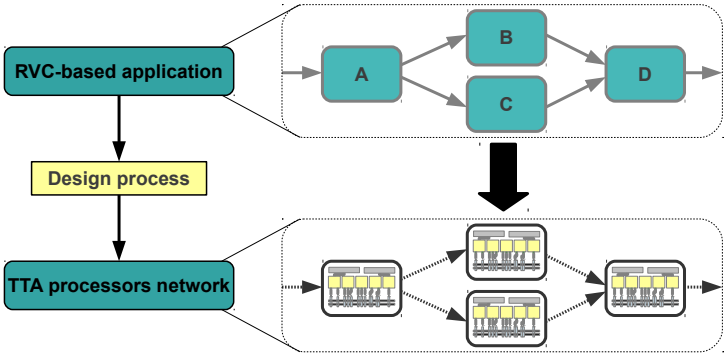

Figure 2: Design approach

Our co-design flow presented in Figure 3 is implemented around two open-source projects: Orcc and TCE. Orcc can be considered as an RVC-CAL front-end for TCE and TCE as a TTA back-end for Orcc. In practice, the design flow is decomposed to network and actor levels. The network level corresponds to the instantiations and interconnections of the components of the design (processors and FIFO channels). This step is performed by Orcc. The actor level is a full co-design flow wherein both TCE and Orcc are involved. Orcc generates a high-level description of the processors and the intermediate representation of the software code associated to each actor, then TCE uses this information to generate a complete processor design: The description of the processor enables the generation of its VHDL description using a pre-existing database of standard hardware components and the software code is compiled into executable binary code.

The processors need to be capable of executing dynamic dataflow programs. RVC programs often consist of actors that have data-dependent behavior, i.e. their execution depends on the value of their input data. To enable this, some specific execution units called *stream units* had to be developed. These *stream units* enable the communication between the concurrently executing processors over hardware FIFO channels. Figure 4 presents an example of an interconnection between two processors. Such stream units have to reproduce the behavior of RVC-CAL FIFOs, particularly their ability to give the number of available tokens and to read data without consuming it (also known as *peeking*).

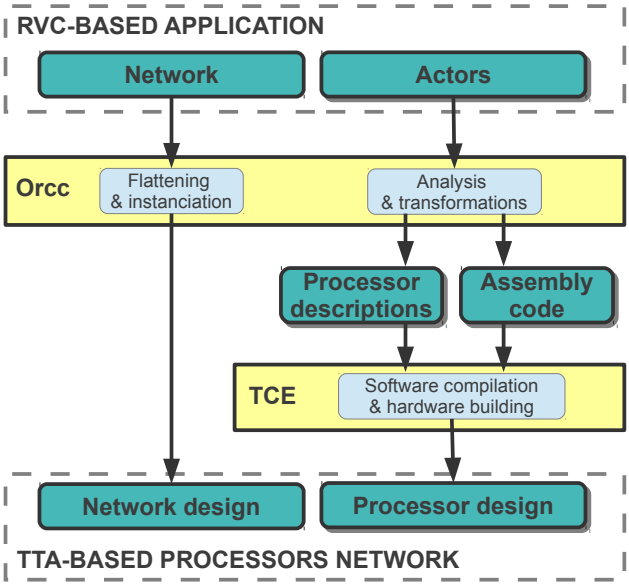

Figure 3: Design flow

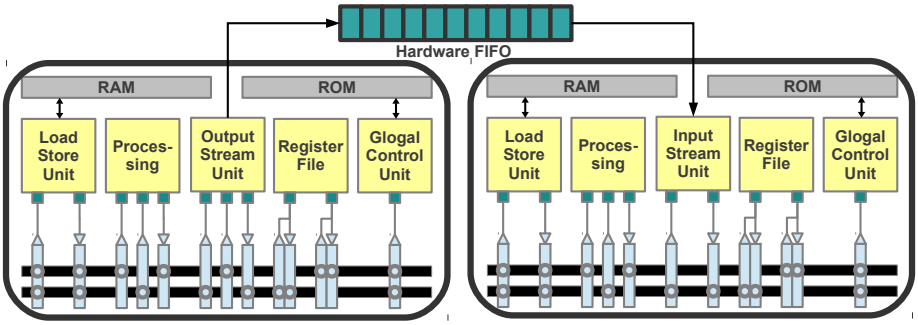

Figure 4: An example of an interconnection between two processors

4.2. Software design flow

The compilation flow is composed of two distinct steps as presented in Figure 5. In the first part, Orcc translates the RVC-CAL code to the intermediate representation. In latter part, performed by the TCE compiler and presented in [7], the intermediate representation is transformed to parallel assembly that is executable by a TTA processor.

The transformation from RVC-CAL code to the LLVM IR was created for this work to enable the proposed design flow. It incorporates several sophisticated transformations and optimizations that are explained below:

1. **Special FIFO operations**: Direct *FIFO read*, *FIFO peek*, *FIFO status* (acquire number of tokens) and *FIFO write* operations are instantiated to

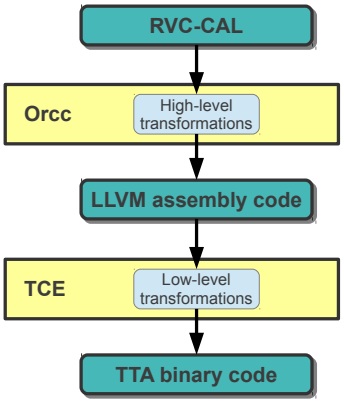

Figure 5: Compilation flow

the LLVM IR. In contrast to, e.g., memory-mapped FIFO access, these special operations allow very fast FIFO communication.

2. **Action scheduler**: In RVC-CAL, the scheduling of actions is expressed using priorities, finite-state machines, guards and constraints on the FIFO states. This transformation expresses the action scheduler in a procedural way to make it understandable by the TTA compiler.
3. **Representation properties**: A total transformation procedure had to be designed to enable making LLVM IR representations out of RVC-CAL actors by respecting properties such as Static-Single Assignment and Three-Address Code. This procedure consists of variable indexing, ϕ-function addition and splitting of complex expressions to multiple primitive instructions.
4. **Correct handling of word lengths**: The RVC-CAL language allows the designer to express bit-accurately the word length of each variable and communication channel. The respective property is also found in the LLVM IR. However, when a computation is to be performed with two variables of different word lengths, the correct result must be ensured by the use of an explicit *cast* instruction, as it is done in the proposed work.

After applying these fundamental transformations, the resulting LLVM IR representation is suitable for the target-independent powerful optimizations of the LLVM compiler then the specific optimizations of the TTA compiler.

4.3. Proposed simulation and debugging infrastructure

Much of the difficulty of adopting MPSoC platforms is due to the following reasons:

- **Debugging** of parallel hardware is very difficult when compared to debugging of software debugging. Execution tracing of hardware blocks is very limited when compared to tracing of software executions.

- **Performance analysis** at platform level is very difficult. Based on the performance of individual blocks, it is impossible to tell anything about the performance of the whole platform.
- **System integration** for MPSoC is a slow and error-prone process.

Our design flow tackles these difficulties by offering a cycle-accurate simulation, using the TCE [7], which can operate at different levels:

- **Actor level:** Each processor can be tested independently from the others. The testing workbench compares automatically produced output data to reference output. A reference output is obtained by running the application on a general purpose processor (for which the C back-end of Orcc generates the software).
- **Network level:** The whole design is simulated to check the functionality of the application including the communication between the processor. This enables evaluating the performance of the application without using an FPGA board.

Moreover, our co-design flow also creates files to enable using a hardware simulator (e.g. Mentor Graphics ModelSim) to check the HDL description. The software simulator is about two hundreds time faster than the hardware one.

5. Experiments

The previous sections presented in detail the automated design flow from RVC-CAL descriptions to a hardware platform composed of a network of TTA processors.

After the generation of such a design, the designer evaluates the performance of the generated hardware components using the simulation tools. If the required performance is not reached, the designer customizes the processors that are identified as bottleneck actors.

In this section, we demonstrate the applicability of our approach using the MPEG-4 Simple Profile video decoder as a case study.

5.1. RVC-CAL description

We have used a description of an MPEG-4 Simple Profile video decoder known as *MVG*, which is available in the Open RVC-CAL Applications bundle at [1]. This description is composed of twenty actors (of which two are simple *broadcasts*), which communicate using forty FIFO channels. These actors are classified as following sub-networks: the *parser* is dedicated to the entropy decoding, *texture* is used to decompress the image texture information and *motion* that performs the motion compensation.

The actors are described at a fine granularity level, most of them compute one block (8x8 pixels) at a time, with very dynamic behavior.

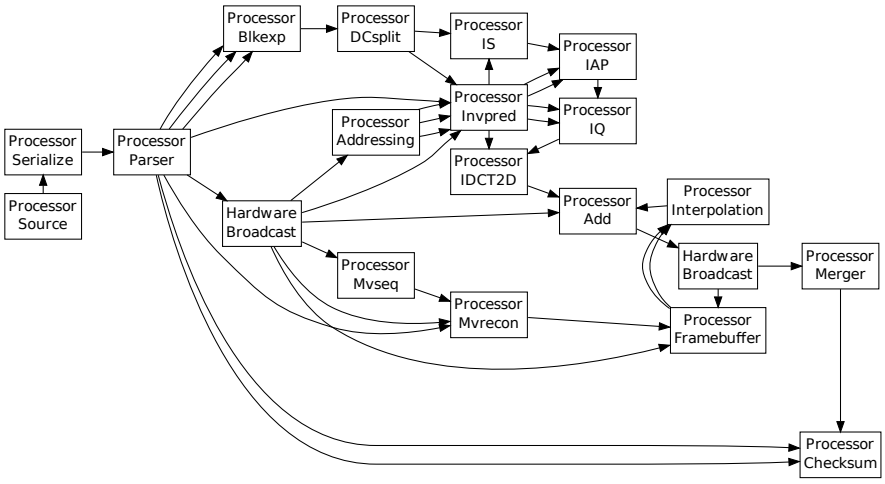

Figure 6: The network of TTA processors that has been synthesized from dataflow description of the MPEG-4 Simple Profile decoder. The arrows between the processors represent the hardware FIFO channels.

The design generated from this description of the MPEG-4 Simple Profile video decoder is presented in Figure 6. This design is composed of 18 processors, two hardware broadcast and 40 hardware FIFOs. The performance evaluation of the generated design has been done by decoding the nine first frames of the *Foreman* sequence (QCIF), available on the website of Orcc.

5.2. Benchmarks

Table 1 describes three different configurations of TTA processors used during the experiments. The first one, called *standard*, is almost equivalent to a RISC processor: inside the TTA processor the interconnection network is composed of two buses that can provide two operands to an execution unit at each clock cycle and move the result when it is available. The two last configurations, *custom* and *huge*, define larger processors composed of several execution units and many buses able to take advantage of the instruction-level parallelism of the application (like a Very Long Instruction Word processor). The *huge* configuration is only used for simulation purposes to acquire the maximal performance.

The simulation results of each actor for the *Standard* and *Huge* configurations are presented in Table 2. These results represent the number of processor cycles needed to produce enough data to decode the given sequence. The simulator assumes that input data are always available during the processor execution. During a real execution, a processor may have to wait for its predecessors. This table shows the maximum performance limited by the application's instruction-level parallelism (ILP).

Unfortunately, the *Huge* configuration can not be used to implement networks of TTA processors on our FPGA boards due to the limited quantity of

Processor	*Standard*	*Custom*	*Huge*
Buses	2	6	32
Arithmetic and logical units	1	4	12
Logical units	1	1	0
Multipliers	1	1	8
Load/Store units	1	1	2
Stream units	2	2	2
Integer register files (32 bits)	2x12	4x12	8x32
Boolean register files (1 bit)	1x2	1x2	1x3
Bus-Unit Interconnection	Full	Full	Full

Table 1: Description of the three different processor configurations used in the experiments

Actor	**Network**	**Standard**	**Huge**	**Speedup**
Invpred	Texture	260000	195000	1,33
Addressing	Texture	484000	415000	1,17
MV sequencing	Parser	530000	338000	1,57
MV reconstruction	Parser	1886000	1560000	1,21
Serialize	-	2153000	1735000	1,24
Inverse scan	Texture	3130000	918000	3,41
DC split	Texture	4337000	3740000	1,16
Parseheader	Parser	5367000	4129000	1,30
Inverse AC pred.	Texture	5370000	2808000	1,91
Block expand	Parser	5399000	4731000	1,14
Inverse quantization	Texture	7014000	3343000	2,10
Merger	-	7325000	2221000	3,30
Interpolation	Motion	7363000	2276000	3,24
Add	Motion	8882000	4186000	2,12
IDCT 2D	Texture	12110000	4648000	2,61
Frame buffer	Motion	15361000	6208000	2,47

Table 2: Simulation results in clock cycles for each actor of a MPEG-4 Simple Profile decoder using two different processor configurations (*Standard* and *Huge*).

available logic. Consequently, we use the smaller *Custom* configuration for the six bottleneck actors and the *Standard* processor for the other ones. Table 3 shows the detailed simulation results for the six bottleneck actors of this RVC decoder. The *Custom* configuration is a good compromise between complexity and performance. This is confirmed by the speedup presented in table 3; it is close to the one acquired with the simulated *Huge* configuration.

The performance results of hardware synthesis are presented in Table 4 for two FPGA boards: Altera Stratix III (EP3SL150F1152C2) and Xilinx Virtex 6

Actor	Network	Standard	Custom	Speedup
Inverse quantification	Texture	7014000	3840000	1,83
Merger	-	7325000	2957000	2,48
Interpolation	Motion	7363000	3037000	2,42
Add	Motion	8882000	4733000	1,88
IDCT 2D	Texture	12110000	6059000	2,00
Frame buffer	Motion	15361000	7646000	2,01

Table 3: Simulation results in clock cycles for six bottleneck actors of a MPEG-4 Simple Profile decoder using two different processor configurations (*Standard* and *Custom*)

(XC6VLX240T). The generated designs dedicated to Xilinx and Altera boards differ only by the proprietary memory components used for RAM, ROM and FIFOs.

FPGA		Standard	Mixed	Ratio
All	Clock Cycles	19800000	9950000	0.5
	FPS (@50MHz)	23	45	2
Altera Stratix III	F_{max} (MHz)	151	106	0.7
	Register	28685	36741	1.3
	Logic	58214	96224	1.6
	RAM (M9K/M144K)	203 / 16	268 / 16	-
	DSP block 18-bit	72	72	-
Xilinx Virtex 6	F_{max} (MHz)	100	91	0.9
	Registers	40251	51286	1.3
	LUTs	58354	90270	1.6
	RAM (B18/B36)	32 / 135	30 / 152	-
	DSP48	60	60	-

Table 4: Hardware synthesis results for a whole MPEG-4 Part 2 Simple Profile decoder using a *Standard* configuration and a mixed (*Standard* and *Custom*) one of the processors network on two different FPGA boards

5.3. Discussion

We have demonstrated the functionality of our design flow by implementing a whole RVC-CAL MPEG-4 Part 2 Simple Profile decoder on two FPGA boards. This application is composed of actors that have very different behavior and granularity. Simulation results with the *Huge* configuration of the processors show two categories of actors, control and computational actors:

1. Control actors have very limited ILP, between 1 and 2 instructions per clock cycle. Their computational needs are minimal or the scheduling of

their actions is too complex to take advantage of the execution parallelism of TTA processors without software strategies like branch predication or hardware mechanisms such as a branch predictor.

2. Computational actors like the inverse discrete cosine transform (IDCT) and interpolation are the traditional bottlenecks of the MPEG-4 SP decoder. The resulting speedup for these actors, between 2.0 and 3.5, depicts their high instruction-level parallelism. They are ideal for execution on VLIW-like processors. This is the reason why it is interesting to use in this case a configuration of TTA processors providing more ILP, like the custom configuration.

The performance on the FPGA board after synthesis shows a speedup of two between the *Standard* configuration and *Custom*. In this particular application, the *buffer* actor remains as the bottleneck that limits the performance of the whole design.

On both FPGA boards, the use of larger processors reduces the maximum clock frequency. Indeed, the critical path of the design increases according to the complexity of the interconnection network in each processor.

In our designs, most of the processors are identical: at most two different TTA processors configurations are used to implement 18 actors. For an optimal performance / resource usage tradeoff, each actor should have a tailored processor. However, the use of identical processors is a first step towards a more generic multi-processor platform able to execute several RVC applications.

6. Conclusion

This paper presents a co-design flow for instantiating many-core systems out of a high-level application description. The many-core system is a network of heterogeneous processors based on the Transport-Trigger Architecture. The presented co-design flow allows a rapid development and evaluation process of complex signal processing applications. We have validated the method with the RVC-CAL description of an MPEG-4 Simple Profile decoder and we are able to automatically generate an FPGA implementation in a few seconds.

The work described in this paper enables future works concerning the dataflow-based design approach of signal processing MPSoCs. The programmability of the TTA processors permits the design of domain-specific platforms able to execute various applications. To reach this target, a generic interconnection model has to be defined.

The platform generation process needs to be improved to reach the performance requirements of modern signal processing systems, such as real-time HD video decoding performance. The application could be accelerated by generating a mixed platform containing some instruction processors and some hardware accelerators generated directly from the RVC-CAL code.

7. Acknowledgments

We would like to thank the following people for their contributions in the Orcc project: Matthieu Wipliez, Antoine Lorence, Khaled Jerbi and Jérôme Gorin. We would also give special thanks to Pekka Jääskeläinen for the time and effort he took to help us with the TCE.

References

[1] Orcc and some RVC-CAL applications are available at the following website. http://orcc.sourceforge.net.

[2] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet. Overview of the MPEG Reconfigurable Video Coding Framework. *Journal of Signal Processing Systems*, 63:251–263, 2011.

[3] J. Boutellier, O. Silvén, and M. Raulet. Automatic synthesis of TTA processor networks from RVC-CAL dataflow programs. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 25–30, 2011.

[4] J. Ceng, J. Castrillon, W. Sheng, H. Scharwachter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda. MAPS: An integrated framework for MPSoC application parallelization. In *45th ACM/IEEE Design Automation Conference*, pages 754–759, 2008.

[5] H. Corporaal. *Microprocessor Architectures : From VLIW to TTA*. John Wiley & Sons, December 1997.

[6] J. Eker and J. W. Janneck. CAL Language Report Specification of the CAL Actor Language. Technical Report UCB/ERL M03/48, EECS Department, University of California, Berkeley, 2003.

[7] O. Esko, P. Jääskeläinen, P. Huerta, C. S. de La Lama, J. Takala, and J. I. Martinez. Customized exposed datapath soft-core design flow with compiler support. In *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications*, FPL '10, pages 217–222, Washington, DC, USA, 2010. IEEE Computer Society.

[8] A. Floch, T. Yuki, C. Guy, S. Derrien, B. Combemale, S. Rajopadhye, and R. B. France. Model-driven engineering and optimizing compilers: a bridge too far? In *Proceedings of the 14th international conference on Model driven engineering languages and systems*, MODELS'11, pages 608–622, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] J. Gorin, M. Wipliez, F. Prêteux, and M. Raulet. LLVM-based and scalable MPEG-RVC decoder. *Journal of Real-Time Image Processing*, 6:59–70, 2011.

[10] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding Sources of Inefficiency in General-Purpose Chips. *Annual International Symposium on Computer Architecture*, pages 85–93, 2010.

[11] J.-M. Hsiao and C.-J. Tsai. Analysis of an SOC Architecture for MPEG Reconfigurable Video Coding Framework. *IEEE International Symposium on Circuits and Systems*, pages 761–764, 2007.

[12] P. Jaaskelainen, E. Salminen, C.S. de La Lama, J. Takala, and J.I. Martinez. TCEMC: A co-design flow for application-specific multicores. In *International Conference on Embedded Computer Systems (SAMOS)*, pages 85–92, 2011.

[13] J. W. Janneck, I. D. Miller, D. B. Parlour, G. Roquier, M. Wipliez, and M. Raulet. Synthesizing hardware from dataflow programs: An MPEG-4 simple profile decoder case study. In *2008 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 287–292, 2008.

[14] K. Jerbi, M. Raulet, O. Déforges, and M. Abid. Automatic Generation Of Optimized And Synthesizable Hardware Implementation From High-Level Dataflow Programs. *VLSI Design*, 2012:Article ID 298396, 2012.

[15] J. Joven, O. Font-Bach, D. Castells-Rufas, R. Martinez, L. Teres, and J. Carrabina. xENoC - An eXperimental Network-On-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architectures. In *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 141–148, 2008.

[16] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland, New York, NY, 1974.

[17] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. page 75. IEEE Computer Society, 2004.

[18] E. A. Lee and T. M. Parks. Dataflow Process Networks. 83(5):773–801, 1995.

[19] L. Nurmi, P. Salmela, P. Kellomäki, P. Jääskeläinen, and J. Takala. Reconfigurable video decoder with transform acceleration. In *2009 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 81–86. IEEE, 2009.

[20] H.-W. Park, H. Oh, and S. Ha. Multiprocessor SoC Design Methods and Tools. *IEEE Signal Processing Magazine*, 26:72–79, 2009.

[21] N. Siret, M. Wipliez, J.-F. Nezan, and A. Rhatay. Hardware code generation from dataflow programs. In *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 113–120, 2010.

[22] A. van Deursen, P. Klint, and J. Visser. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35(6):26–36, 2000.

[23] M. Wipliez, G. Roquier, M. Raulet, J.-F. Nezan, and O. Déforges. Code generation for the MPEG Reconfigurable Video Coding framework: From CAL actions to C functions. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 1049 – 1052, 2008.

[24] W. Wolf, A.A. Jerraya, and G. Martin. Multiprocessor System-on-Chip (MPSoC) Technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1701–1713, 2008.